FUNDAMENTOS DE PROGRAMACIÓN Convocatoria: **Septiembre** Semana: **Nacional / U.E.** Tipo de examen: **A**

El **test** debe ser contestado en la **hoja de lectura óptica**. Sólo una de las cuatro respuestas posibles de cada pregunta es correcta.
El test es eliminatorio y aporta un 30% de la nota final. Son necesarias 8 preguntas correctas (6 con las prácticas aprobadas) para que se corrija el ejercicio.
Cada respuesta correcta: 1 punto. Respuesta incorrecta o en blanco: 0 puntos.

1. Dada la siguiente función en C±:

```
float suma(float x){
  if(x == 5.0){ return 1.0/x; }
  else{ return((1.0/x)+suma(x+1.0)); }
}
```

   A. La llamada `suma(1.0)` devuelve `0.0`
   B. La llamada `suma(1.0)` devuelve `1.0`
   C. La llamada `suma(1.0)` devuelve `1.563333`
   D. La llamada `suma(1.0)` devuelve `2.283333`

2. Cuando se utiliza:

```
do { Acción } while ( Condición );
```

   A. `Acción` y `Condición` se ejecutan de `1` a `N` veces
   B. `Acción` se ejecuta de `0` a `N` veces y `Condición` de `1` a `N` veces
   C. `Condición` se ejecuta de `0` a `N` veces y `Acción` de `1` a `N` veces
   D. `Acción` y `Condición` se ejecutan de `0` a `N` veces

3. Dada la descripción formal de un programa por la regla de producción:

   Programa ::= **{** *Include* **}** `int main()` *Bloque*

   Podemos afirmar que:

   A. *Include* y *Bloque* son elementos terminales
   B. `int main()` es un elemento terminal
   C. `int main()` y *Bloque* son elementos no terminales
   D. `int main()` e *Include* son elementos terminales

4. El siguiente código en C±:

```
int x = 10;
do {
  printf("%d ", x);
  x = x*2/3-1;
} while(x%3 > 0);
```

   A. Imprime solamente un `10`
   B. Imprime una cantidad infinita de `10`
   C. Imprime `10 5`
   D. Imprime `10 5 2`

5. En C±, la ejecución de una sentencia `throw`:

   A. Realiza el tratamiento de la excepción
   B. Evalúa si se ha producido una excepción
   C. Siempre debe estar condicionada
   D. Efectúa la programación a la defensiva

6. El valor que determina la selección en la sentencia switch de C± no puede ser de tipo:

   A. `int`
   B. `float`
   C. `Enumerado`
   D. `char`

7. En C±, cuando se utiliza:

```
typedef struct Uno {Dos Tres; Cuatro Cinco;};
```

   A. `Tres` y `Cinco` pueden ser el mismo identificador
   B. `Uno` y `Cinco` pueden ser el mismo identificador
   C. `Dos` y `Cuatro` pueden ser el mismo identificador
   D. `Uno` y `Tres` pueden ser el mismo identificador

8. Señale cuál de las siguientes afirmaciones es falsa:

   A. En C±, `#include` es una función predefinida
   B. En C±, la función `isalpha(c)` indica si `c` es una letra
   C. C± soporta la definición de nuevas funciones
   D. C± soporta la definición de nuevos tipos de datos

9. El fragmento de programa:

```
z = y;
```

   A. Es una expresión booleana
   B. Equivale a `y = z;`
   C. `z` e `y` deben ser números
   D. Es una sentencia

10. Dado el siguiente código en C±:

```
#include <stdio.h>
int F(int a, int b){
  a = 2; b = 10; return 0;
}
int main(){
  int b,a,t;
  a = 10; b = 11; t = F(a,b);
  printf("%d,%d", a,b);
}
```

    ¿Qué valores se muestran?

    A. 2, 10
    B. 10, 2
    C. 10, 11
    D. 11, 10

EJERCICIO DE PROGRAMACIÓN

Realizar en C± el TAD, con fichero de interfaz y de implementación, **Matriz7**, el cual emplea un tipo de datos Vector7 para manejar matrices 7x7 de valores enteros. Los subprogramas a realizar son: **SumarFilas**, que devuelve un vector cuyos elementos son la suma de las filas de la matriz de entrada (el primer elemento del vector contendrá la suma de los elementos de la primera fila de la matriz, el segundo elemento del vector contendrá la suma de los elementos de la segunda fila de la matriz y así sucesivamente); **TransponerMatriz**, que devuelve la matriz transpuesta ($a_{ij} = a_{ji}$ para todo $i, j = 1, \ldots, 7$); **EsEscalar**, que devuelve cierto si la matriz es escalar (una matriz es escalar si todos los elementos que no son de la diagonal principal son ceros y $a_{ii} = a_{jj}$ para todo $i, j = 1, \ldots, 7$) y falso en caso contrario.